命令型プログラミングにおける動作理解支援に関する研究

2011SE033 長谷川洸也　2011SE122 川地周作
指導教員：蜂巣吉成

1　はじめに

現在，情報系の大学などでは，プログラミングの学習が行われている．初めてプログラムにふれる学習者が，プログラムの動作を理解していない状態でプログラミングをすることは難しい．また，実行結果のみを見て学習を進めていく学習者も存在し，プログラムの状態の変化を理解していない可能性がある．プログラミング学習で用いられる C 言語などの命令型言語では，命令文を実行し，プログラムの状態である変数の値の集合を変化させることで計算が行われる．プログラミング学習では，プログラムの状態が学習者の意図にあった状態に変化しているかを理解することが重要である．

初学者がプログラミングを行う際，ソースコードの書き間違いや理解不足による文法の間違いがあると，コンパイルエラーが表示されるのでプログラムがどこで間違っているかを理解することができる．しかし，コンパイルエラーが出力されていないのにも関わらず，初学者の意図した結果になっていない場合がある．その場合，初学者が実行結果だけを見て，どこで間違いが起こっているのかを把握することやなぜ間違った動作をしているのかを理解することは難しく，プログラムを修正するためには，プログラムの状態を理解することが必要になる．また，初学者の意図した結果になっていたとしても，プログラムの状態を理解することは重要である．動作理解には，デバッガやソースコード中に printf 文を使い変数の変化を確認する方法があるが，初学者は変数を確認するタイミングが分からないので利用する者は多くない．

プログラミング学習における動作理解支援を目的としたシステムがいくつか提案されている．[1]，[2] では，学習者がソースコードを逐次実行させながらその都度変化する変数の値を確認することができる．しかし，これらの方法では，現在どのような動作を行っているかを学習者に表示することはできるが，過去の動作記録を表示していないので，現在の値とそれまでの値を比較することができない．また，プログラム全体の動作を表示していないので，学習者が見たい箇所まで逐次実行をする必要がある．

そこで本研究では，変数の値の変化を可視化させることにより，初学者のプログラムの動作理解を支援するツールを提案する．C 言語などの命令型言語では，変数の値を書き換える命令文を記述することによってプログラムを作成していく．本研究では，制御構造の観点から，連接，分岐，繰り返し，データ構造の観点から，基本型，配列を可視化する方法を提案する．基本型，配列を可視化するために，変数の値の集合を命令文の横に表示する．分岐を理解させるために，条件式の真偽を表示する．繰り返しを理解させるために，繰り返し文での変数の変化を全て可視化する．変数の変化や条件式の真偽を可視化させることで，プログラムの動作を視覚的に確認し理解することができると考える．本研究では，C 言語を対象とし，初学者とはプログラミング学習者の中でも，初めてプログラミングにふれる学習者とする．初学者の書くプログラムは十数行とし，変数の数も少ないものとする．また，繰り返し文は 2 重までとし，繰り返し回数も数十回程度とする．関数，ポインタの可視化についても考察する．

本研究の目的は，命令型プログラミング言語の動作理解支援であるが，学習者がツールを使うことにより，プログラムの誤り発見やデバッガの利便性についても理解できるようになる．

2　関連研究

古宮らは，プログラミング初学習者を対象としたプログラミング学習支援環境 (AZUR) を開発し，学習者が陥りやすい誤りを防ぐために可視化している [1]．また，このツールでは逐次実行が可能で，実行中の行を追いながら各変数の値の変化も確認できる．

佐藤らは，プログラムの動作を可視化するために，変数や計算過程を図を用いて表現している [2]．このツールでは，逐次実行を行うことで実行中の行を追いながら各変数の値の変化や入力，出力を図で確認できる．

[1]，[2] の研究は，いずれも完成されたプログラムの動作を理解することに有効であるが，学習者の意図していない実行結果の場合，その間違いを見つけることには不向きである．また，いずれも逐次実行を行い，プログラムをたどりながら現在どのような動作を行っているかを学習者に理解させようとしているが，[1] では，プログラムをたどっている最中に現在までの動作と現在行っている動作の比較をすることができず，プログラム全体の動作理解を支援することができない．[2] では，現在逐次実行している行の 1 つ前に戻る逆実行機能により，現在までの動作を繰り返し確認することができるが，変数の数が多くなればなるほど可視化部分が見にくくなってしまう欠点がある．また，これらの研究では，回数が多い繰り返しがプログラム内に存在する場合，逐次実行によってプログラムをたどるには手間がかかる．

GDB[3] や Eclipse[5] のようなデバッガを初学者に使わせる方法もあるが，新たにコマンドを覚えないといけないことや設定に時間がかかる．ブレークポイントの設定や逐次実行で確認すべきタイミングが分からないと，初学者にデバッガを使わせることは不向きである．

本研究では，プログラム全体の動作を可視化させることで，デバッガにはない一覧性を持たせることができる．

3　動作理解支援方法

3.1　命令型プログラミング言語の動作理解

命令型言語を理解するためには，データ構造の観点より，プログラムの状態である変数の変化を各命令文毎に確認する必要がある．制御構造の観点より，分岐文によりどの命令文を実行したか，繰り返し文により，何回命令文を実行したか，各繰り返し文においてどのように状態が変化しているかを確認することが必要である．

3.2　命令型プログラミング言語の支援方法

3.1 節であげた命令型言語を理解するために必要なことを，学習者にどのように支援させるかを示す．

- 各命令文毎に変数の変化を確認できるようにする
  1. 代入による変数の値の変化を確認できるようにする
- 条件文について
  2. 条件式の真偽を確認できるようにする
  3. 条件文により，どの命令文が実行されているか確認できるようにする
- 繰り返し文について
  4. 何回繰り返されるかを確認できるようにする
  5. 一回の繰り返しにおいて，どのように状態が変化し，計算を行うか確認できるようにする
  6. 各回の繰り返しにて，変数の値の変化を確認できるようにする
  7. 繰り返しを続けるための条件の真偽がどのように変わるかを確認できるようにする

これらを表示するために，本研究ではプログラムの動作理解支援ツールを提案する．

4　動作理解支援ツールの設計と実現

4.1　設計

ツール内部で，GDB[3] を用いてソースコードを逐次実行することにより，プログラム内の変数を取得，記録する．記録した値の変化を表示する事で，初学者にプログラムの動作理解支援を行う．変数の変化の値の表示について，プログラム全体の値を確認できる表での表示を提案する．3.2 節であげた 1 を支援するために，ソースコードの各行ごとに全ての変数の値を表示することで，利用者が確認したい行の変数の値と他の行の変数の値を比較をすることができる．また，既存のツール同様，ソースコードの行数や変数の数が増えると表として表示するには見づらくなっていくが，本研究では，初学者を対象としていること，関数を定義しない main 関数のみのプログラムでは，ソースコードは数十行程度，変数の数も多くて十数個程度であることより，表にしても問題がないと考える．条件文について，3.2 節であげた 2，3 を支援するために，条件式の真偽を TRUE か FALSE かを表示する．実行されない行はソースコードの色を薄く表示させることで，どの行が実行されたかを示す．繰り返し文について，繰り返し回数分，カラムを増やすことで，各繰り返し毎にどのように変数の値が変化しているのかを確認できると考えた．これにより，3.2 節であげた 5，6 を支援することができる．また，繰り返し回数分，カラムを増やすことで繰り返し回数を確認することができ，4 を支援することができる．分岐分の表示方法と同様に，繰り返しを続けるための条件の真偽を表示することで，7 を支援することができる．表をすべて表示させることで，各繰り返し文毎に変化を確認できる．対象が初学者であるので，繰り返し回数は多くても数十回程度であることより，表を全て表示しても問題がないと考える．

本研究で開発するツールでは GDB を用いる．支援ツールを作成するにあたり，初学者の書いたソースコードを自動的に逐次実行したり変数の取得をしたりするために，GDB を用いることが効率的であると考えた．GDB はローカル変数を取得した時に，配列，構造体の中身が全て表示され，print コマンドを使用することにより，変数のアドレスの表示や条件式の真偽を確認する事が可能である．また，GDB では自動的に 0 が代入されず，変数が初期化されないので，初期化忘れが原因の誤りを発見する支援が可能となる．

初学者の書いたソースコードから変数の値を取得する方法として，デバッガやインタプリタを使わずに初学者の記述するソースコードに変数を表示する printf 文を追加し，実行結果の取得と同時に変数の値を取得する方法がある．しかし，この方法では，初学者の書くソースコードに新たに命令文を追加するので，本来ある命令文の行数が変わり，行数が正しく取得できない場合があると考えた．

4.2　実現

本ツールの処理の流れを図 1 に示す．図 1 の source.c はコンパイルエラーのない C 言語のソースコードである．本ツールは Java を用いて開発し，表示には Java の GUI ツールキットである Swing を利用して変数の値を表で表示している．

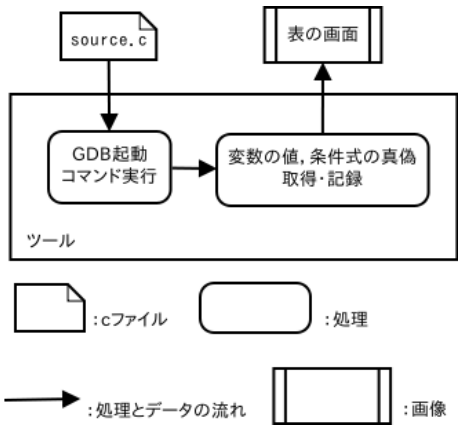
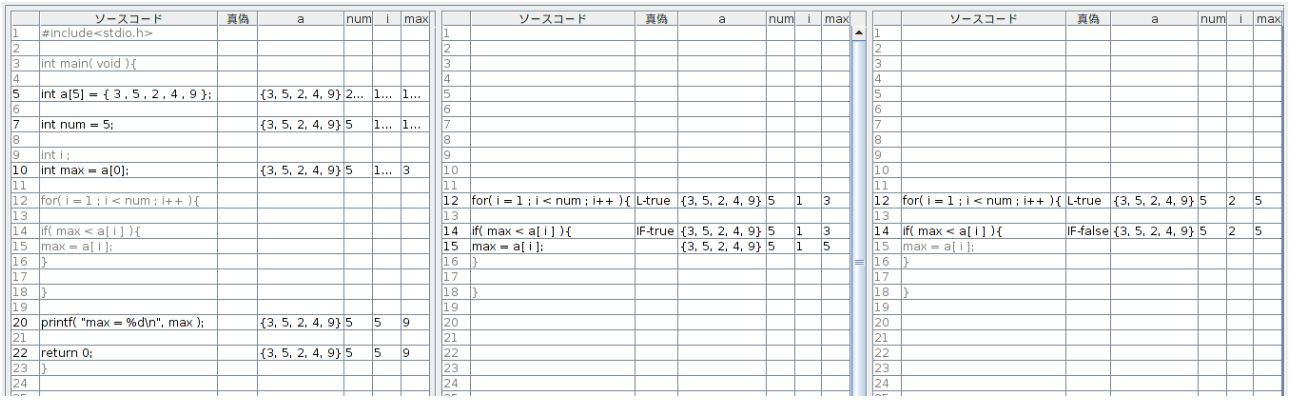


図 1　ツール内の処理



	ソースコード	真偽	a	num	i	max
1	#include<stdio.h>					
2						
3	int main(void){					
4						
5	int a[5] = { 3 , 5 , 2 , 4 , 9 };		{3, 5, 2, 4, 9}	2...	1...	1...
6						
7	int num = 5;		{3, 5, 2, 4, 9}	5	1...	1...
8						
9	int i ;					
10	int max = a[0];		{3, 5, 2, 4, 9}	5	1...	3
11						
12	for(i = 1 ; i < num ; i++){					
13						
14	if(max < a[i]){					
15	max = a[i];					
16	}					
17						
18	}					
19						
20	printf("max = %d\n", max);		{3, 5, 2, 4, 9}	5	5	9
21						
22	return 0;		{3, 5, 2, 4, 9}	5	5	9
23	}					
24						

	ソースコード	真偽	a	num	i	max
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12	for(i = 1 ; i < num ; i++){	L-true	{3, 5, 2, 4, 9}	5	1	3
13						
14	if(max < a[i]){	IF-true	{3, 5, 2, 4, 9}	5	1	3
15	max = a[i];		{3, 5, 2, 4, 9}	5	1	5
16	}					
17						
18	}					
19						
20						
21						
22						
23						
24						

	ソースコード	真偽	a	num	i	max
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12	for(i = 1 ; i < num ; i++){	L-true	{3, 5, 2, 4, 9}	5	2	5
13						
14	if(max < a[i]){	IF-false	{3, 5, 2, 4, 9}	5	2	5
15	max = a[i];					
16	}					
17						
18	}					
19						
20						
21						
22						
23						
24						

図 2　ツール画面

ツールを起動すると図 2 のような表が表示され，表の列は左から順に行数，ソースコード，条件式の真偽，各変数の値の変化が表示されている．行はソースコードの行数に対応している．真偽の列には，if 文の条件式の真偽，for 文，while 文の条件式の真偽が表示される．else-if 文が連続する場合，真偽を判定したもののみ真偽を表示するようにした．これにより，図 3 のようにどの条件式までを判定したか分かるようになる．

5	int a = 4;		4
6			
7	if(a < 1){	IF-false	4
8	printf("a<1");		
9	}		
10	else if(a < 3){	IF-false	4
11	printf("a<3");		
12	}		
13	else if(a < 5){	IF-true	4
14	printf("a<5");		4
15	}		
16	else if(a < 7){		
17	printf("a<7");		
18	}		
19	else{		
20	printf("a>=7");		
21	}		

図 3　条件文の真偽のツール画面

各変数の値の変化は，命令文が記述されている行のみ表示され，その行の命令文が実行された後の変数の変化を表示している．ソースコード内に繰り返し文が見つかると表の右隣に新たにカラムが追加される．追加されたソースコードのカラムには，繰り返し文により繰り返される範囲のみのソースコードが記述される．変数の数や繰り返しにより表示されるカラムが増えると表示しきれない場合があるが，表の下の水平方向のスクロールバーを移動させることで表示しきれない変数や表を確認することができる．ソースコード内に 2 重の繰り返し文があると，図 4 のように 1 重目の繰り返し文の表の途中に 2 重目の繰り返し文が並んで表示される．この表示方法により，ツール利用者はカラムを順に確認する事で 2 重目の繰り返し文中の変数の値の変化や実行される命令文を確認できると考えた．ソースコード内に scanf 文があると入力を求めるメッセージボックスが表示され，実際のシェルに入力する書き方で入力が可能である．

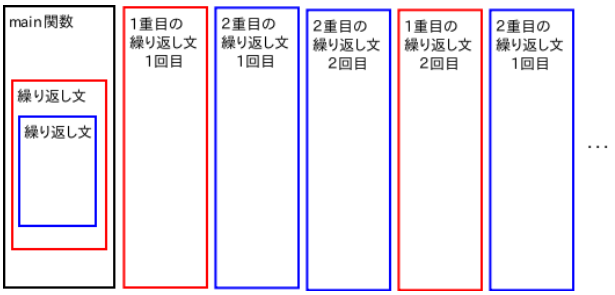


図 4　繰り返し文のツール画面

5　評価

5.1　評価方法

本研究で提案したツールをプログラミング演習を履修済みの大学 3 年生に利用してもらい，ツールを使って動作理解支援が行えるか確認する．学生 10 名をツールを使うグループと使わないグループに分け，5.2 節で示す 3 問を解いてもらい，正答数の比較やアンケート結果から誤りが訂正できるかを確認する．いずれも，コンパイルエラーはないが誤りがあるプログラムである．また，両グループともソースコードを編集し，コンパイル，実行することができるものとする．

5.2　問題

- 問 1　バブルソートを訂正させる問題
- 問 2　右下が直角な直角二等辺三角形を作るプログラムを訂正させる問題
- 問 3　文字列の大文字と小文字を入れ替えコピーさせるプログラムを訂正させる問題

ソースコード 1 は，問 3 の文字列 str1 の大文字と小文字を入れ替え，str2 にコピーするプログラムの一部である．このプログラムの誤りは，8 行目が else-if でなく if 文になっている点である．図 5 は，ソースコード 1 のツール画面の一部である．

ソースコード 1　問 3 のプログラムの一部

```
1  for( i = 0 ; str1[i] != '\0' ; i++ ){
2
3          str2[i] = str1[i];
4
5          if( 'a' <= str2[i] && str2[i] <= 'z' ){
6                  str2[i] = str2[i] - 'a' + 'A';
7          }
8          if( 'A' <= str2[i] && str2[i] <= 'Z' ){
9                  str2[i] = str2[i] - 'A' + 'a';
10         }
11
12 }
```

10	for(i = 0 ...	L-true	"Hello World", ...	"horld Wide Web"...	1
11					
12	str2[i] = ...		"Hello World", ...	"herld Wide Web"...	1
13					
14	if('a' <= ...	IF-true	"Hello World", ...	"herld Wide Web"...	1
15	str2[i] = ...		"Hello World", ...	"hErld Wide Web"...	1
16	}				
17	if('A' <= ...	IF-true	"Hello World", ...	"hErld Wide Web"...	1
18	str2[i] = ...		"Hello World", ...	"herld Wide Web"...	1
19	}				
20					
21	}				
22					

図 5　問 3 ツール画面

5.3　評価結果の分析

評価の結果，ツール使用者の正当数が若干上回る程度であった．しかし，問 3 において，全て訂正できた人は，ツール未使用者の正解者が 1 人に対してツール使用者は 4 人であった．図 5 のツールの表を見ても分かる通り，一つ目の分岐文の条件を満たした場合，文字が大文字に変換され，その変換された文字が二つ目の分岐文の条件を満たしてしまっていることが分かる．これは，繰り返し文をすべて表示したことと，条件式の評価を表示し，分岐文により実行される命令文と実行されない命令文を色分けして表示したことにより誤りを発見することができたと考える．

本研究では，初学者向けのツールとして作成してきたので，評価の結果より初学者にも効果はあると考える．

6　考察

本研究では分岐文や繰り返し文などの初歩的な構文を対象としている．理由として，ポインタや関数を学ぶ前の段階で躓く学生が存在し，彼らの一部はプログラムの動作を理解しないまま，次の段階へ進んで行くということが起きているからである．しかし，C 言語の動作理解におけるポインタや関数は学生の多くが理解不足になる点でもある．本ツールでは，GDB を用いているので，各変数のアドレスを取得することが可能である．表にアドレスを表示することで，ポインタが指すアドレスがどの変数かが分かる．しかし，アドレスを表示しただけでは理解することが難しいので，分かりやすくするために同じアドレスを対応させ，背景を同じ色にすることや，ポインタが指す変数名を表示させるなどの改善が必要であると考える．関数について，本ツールでは命令文の隣に変数の値の変化を表示しているので，関数が何度も呼ばれた場合，別の表に表示してもその関数がいつ呼ばれたものか分かりづらい．関数の動作理解を行うためには，表以外の方法で可視化する必要があると考える．

本研究では，初学者の書くプログラムが十数行程度であり，変数の数も少ないと仮定しているが，ソースコードが長くなるにつれ変数が多くなり初学者が着目したい変数の変化を確認しづらくなる問題が発生する．プログラムスライシングを用いて，プログラム内の任意の文の着目する変数に影響を与える一部のコードや変数の値を強調して表示させることで，着目する変数の変化が分かりやすくなる．繰り返し文の表示方法について，本研究では，全ての変数の変化を表示しているので，繰り返し文内に出現しない変数の値も表示している．繰り返し文内で出現しない変数を省略することで，利用者が着目する命令文で，どの変数が変化しているかを把握しやすくなる．

7　おわりに

本研究では，初学者のプログラミング学習において実行結果のみならず，プログラミングの動作理解も重要であることから，プログラミング学習における動作理解の支援を実現するツールを提案した．GDB を用いてソースコードを逐次実行し，変化する変数の値を表で表示した．評価結果より，本ツールが動作理解に適していることを示した．

今後の課題としては，ポインタや関数の表示方法の検討，実現を行う必要がある．また，全ての変数の変化の値を表示させているので，初学者が着目した変数の値を確認しやすい表示方法を実現することが今後の課題である．

参考文献

[1] 古宮誠一，今泉俊幸，橋浦弘明，松浦佐江子：『プログラミング学習支援環境 AZUR ーブロック構造と関数動作の可視化による支援ー』情報理学会研究報告，ソフトウェア工学研究会，Vol.2014-SE-183，No.5，pp.1-8，2014.

[2] 佐藤美奈，安井浩之，横山孝典：『プログラムの動作を可視化する教育用プログラミング言語環境の提案』情報処理学会研究報告，コンピュータと教育研究会報告，Vol.2009-CE-102，No.19，pp.1-6，2009.

[3] 公式サイト『GDB: The GNU Project Debugger』http://www.gnu.org/software/gdb/.

[4] Automating gdb - Piping stdin and stdout for gdb，http://www.solutionoferror.com/java/automating-gdb-piping-stdin-and-stdout-for-gdb-146394.asp.

[5] 公式サイト『eclipse』，http://www.eclipse.org/.

[6] ラビ・セシィ：『プログラミング言語の概念と構造 [新装版]』，株式会社ピアソン・エデュケーション，2002.